Theoretical Computer Science 48 (1986) 153-181
North-Holland

HIERARCHIES OF ONE-WAY MULTIHEAD AUTOMATA LANGUAGES *

Marek CHROBAK

Institute of Informatics, Warsaw University, PKiN VIIIp, 00-901 Warsaw, Poland

Communicated by G. Mirkowska
Received April 1985
Revised August 1986

Abstract. Let DPDA(k) (respectively NPDA(k)) be the class of languages recognized by one-way k-head deterministic (respectively nondeterministic) pushdown automata. The main result of this paper is that, for every $k>0$, DPDA(k) $\subsetneq$ DPDA($k+1$) and DPDA(k) $\subsetneq$ NPDA(k).

1. Notation

We will consider the following automata:

- dfa(k): a one-way k-head deterministic finite automaton,
- nfa(k): a one-way k-head nondeterministic finite automaton,
- dca(k): a one-way k-head deterministic counter automaton,
- nca(k): a one-way k-head nondeterministic counter automaton,
- dpda(k): a one-way k-head deterministic pushdown automaton,
- npda(k): a one-way k-head nondeterministic pushdown automaton,
- 2dfa(k): a two-way k-head deterministic finite automaton,
- 2nfa(k): a two-way k-head nondeterministic finite automaton,
- 2dca: a two-way deterministic counter automaton (with one head),
- 2nca: a two-way nondeterministic counter automaton,
- 2dpda(k): a two-way k-head deterministic pushdown automaton,
- 2npda(k): a two-way k-head nondeterministic pushdown automaton.

If $x(k)$ is a type of automaton, then $X(k)$ is the class of languages recognized by the $x(k)$ automaton. So, for example, 2NFA(3) is the class of languages recognized by 2nfa(3) automata. When k is not specified, then $k=1$ is assumed. For example, 2dpda is a one-head two-way deterministic pushdown automaton.

By h_i we will denote the ith head of the automaton under consideration. If ambiguity arises, we will use h_i and h_i', or superscripts.

By REG we denote the class of regular sets: REG = DFA = NFA. It is well known also that 2DFA = 2NFA = REG.

* A preliminary version of this paper appeared in *Proc. 12th Internat. Coll. on Automata, Languages and Programming*, Lecture Notes in Computer Science **194** (Springer, Berlin, 1985).

A language L is said to be strictly n-bounded if there are n distinct symbols $a_1, a_2, \ldots, a_n$ such that $L \subseteq a_1^* \ldots a_n^*$. L is strictly bounded if it is n-strictly bounded for some n. Since we will consider only strictly bounded languages, we will simply say 'bounded' instead of 'strictly bounded'.

2. Introduction

Given a class of multihead automata, the following problems are usually investigated:

(1) are $k+1$ heads better than k?

(2) are nondeterministic automata better than deterministic ones?

(3) closure properties.

These problems were already considered for many types of automata. The first definition of multihead automata appeared in the early 60's in Piatkowski [35]. There he defined a dfa(k) and soon after, Rosenberg [36, 37] investigated problems (1)-(3) for dfa(k)'s. Unfortunately, the proofs in [37] were incorrect (see [6]). The first progress (for one-way automata) was made in 1975 when Ibarra and Kim [24] proved that $\mathrm{DFA}(2) \subsetneq \mathrm{DFA}(3)$. These classes were separated by the language $\{a^i b^j c^k : i=j \text{ or } j=k \text{ or } k=i\}$. A different proof of this inequality was given by Sudborough [40].

The full solution to (1) for one-way automata was presented in 1978 by Yao and Rivest [42], who proved that $\mathrm{DFA}(k) \subsetneq \mathrm{DFA}(k+1)$ and $\mathrm{NFA}(k) \subsetneq \mathrm{NFA}(k+1)$ for every $k>0$. They also proved that $\mathrm{DFA}(k) \subsetneq \mathrm{NFA}(k)$ for $k>1$. In fact, $\mathrm{NFA}(2) - \bigcup_{k>0} \mathrm{DFA}(k) \neq \emptyset$. The proof was by a clever counting argument, using some observations of Rosenberg [37]. Their technique (often called 'cutting and pasting' or 'fooling') was applied by Miyano [27] to prove similar results for counter machines: $\mathrm{DCA}(k) \subsetneq \mathrm{DCA}(k+1)$, $\mathrm{NCA}(k) \subsetneq \mathrm{NCA}(k+1)$ for every $k>0$. Hromkovič [19] used suitably modified cutting and pasting to prove some non-closure properties of DFA(k). Problems (1)-(3) were also considered in [4, 25] for so-called simple multihead automata, that is, for such automata that only one head sees the input symbols, the other $k-1$ heads can detect only the endmarker.

Allowing the heads to move in both directions we obtain appropriate two-way automata [11, 13, 22]. In contrast to one-way devices, two-way ones are not looked upon as just another abstract model of computation. The importance of two-way automata stems from the fact that they often define some known complexity classes [2, 14, 21]. For example,

$$\textsc{Dlog} = \bigcup_{k>0} \mathrm{2DFA}(k), \qquad \textsc{Nlog} = \bigcup_{k>0} \mathrm{2NFA}(k),$$

$$\textsc{Ptime} = \bigcup_{k>0} \mathrm{2DPDA}(k) = \bigcup_{k>0} \mathrm{2NPDA}(k).$$

Also some complexity problems as P = NP and LBA, can be reduced to simple-looking problems about two-way automata [7, 14, 30], even those accepting only 1-bounded languages [29].

Two-way automata are much more powerful than one-way ones. It may then be surprising that problem (1) was first solved for two-way automata. Fortunately, they are powerful enough to allow application of two frequently used techniques for proving separation results: diagonalization and padding. Using them Ibarra [22] proved that, for every $k>0$,

$$2DFA(k) \subsetneq 2DFA(k+2), \qquad 2DPDA(k) \subsetneq 2DPDA(k+1),$$

$$2NPDA(k) \subsetneq 2NPDA(k+1).$$

A weaker result appeared in Hartmanis [14]: $2DFA(k) \subsetneq 2DFA(k+3)$ for $k>0$. The full hierarchy result for finite automata was proved by Monien [30], who showed that

$$2DFA(k) \subsetneq 2DFA(k+1) \quad \text{and} \quad 2NFA(k) \subsetneq 2NFA(k+1) \qquad \text{for } k>0.$$

In the mid 70's one may observe a tendency to separate complexity classes using possibly simple languages [6, 15, 38, 39], namely 1-bounded languages. This approach was followed by Seiferas [39] and Sudborough [41] who proved that, for $k>0$,

$$2DFA(k) \cap B_1 \subsetneq 2DFA(k+4) \cap B_1,$$

$$2NFA(k) \cap B_1 \subsetneq 2NFA(k+4) \cap B_1.$$

Monien [31] closed the gaps in their results by proving that

$$2DFA(k) \cap B_1 \subsetneq 2DFA(k+1) \cap B_1,$$

$$2NFA(k) \cap B_1 \subsetneq 2NFA(k+1) \cap B_1.$$

As already said, all these hierarchies were obtained by diagonalization. For comparatively simple automata and low-level complexity classes it is possible to prove separation results without using diagonalization. For example, Ďuriš and Galil [3] invented an ingenious counting argument to show that $2DCA \subsetneq 2DPDA$. Their method was used in Chrobak [1] to prove that $2DCA \subsetneq 2NCA$. Except well-known crossing sequence arguments [6, 17], there are two more new techniques which should be mentioned here. The first one is the information-theoretic approach based on the so-called Kolmogorov complexity of strings [26, 33]. It was used to show lower bounds for the simulation of $(k+1)$-tape TM's by k-tape TM's [26, 33], and lower bounds for string-matching [26]. The second technique, introduced by Paul, Pippenger, Szemerédi, and Trotter [34] to prove that $\text{DTIME}(n) \subsetneq \text{NTIME}(n)$, is a graph-theoretic approach. Their result was derived from some properties of so-called Turing machine graphs.

The reader might have noted that we have not yet mentioned one-way multihead pushdown machines. The reason is that except the $k=1$ case, all problems (1)–(3)

for these automata are still open. Multihead pushdown automata seem too powerful to fool them using counting arguments. Observe that an nfa(k) has at most $O(n^k)$ configurations. The proof of Miyano [27] was possible because an nca(k) has $O(n^{k+1})$ configurations, that is, still polynomially many. But an npda(k) may have $O(2^n)$ configurations because the number of possible contents of the pushdown store is exponential. This makes the application of counting arguments to fool an npda(k) rather unlikely.

Also the semilinear property of npda(k) languages [20, 23] is of no use to us; it can be applied only to separate the whole class $\bigcup_{k>0}$ NPDA(k) from other classes of languages not possessing this property.

These problems were partially overcome by Miyano [28], but his definition of multihead automata differs from the one in [13]: the input tape has no endmarkers and the automata accept by entering an accepting state when one of the heads falls off the tape. He proved that for these machines $k+1$ heads are better than k. The proof is by reduction to the analogous problem for two-way automata, so, in essence, it is a proof by diagonalization.

In this paper we introduce a new technique for proving separation results which can be looked upon as a refinement of the so-called cycle technique. The cycle technique is based on the following observation.

Suppose that a dfa A has a^m on the input, where m is greater than the number of states of A. During the computation A must eventually enter a cycle, and let s be the number of A's moves in this cycle. Then A accepts a^m iff A accepts a^{m+s}. If A is a dfa(k), $k>1$, then the situation is more complicated. In fact, so far, the cycle technique was used only for $k=2$. We investigate the behaviour of A on n-bounded inputs, where n is possibly small. Usually, the witness language is chosen to be at most 4-bounded. For such inputs we can consider only 'boundary configurations', that is, such configurations in which one of the heads enters a new block of symbols on the input. It is important here to note that for fixed k the number of boundary configurations of a dfa(k) on n-bounded inputs is at most nk. The positions of the heads in consecutive boundary configurations are related by some linear equations and the coefficients in these equations depend only on the description of A. An investigation of these relations (usually by considering numerous cases) allows us to fool A by pumping the input and forcing A to accept a word which should not be accepted.

The cycle technique was used by Ibarra and Kim [24] to separate DFA(2) and DFA(3). Note that their witness language is 3-bounded. Hromkovič [18] also used similar ideas to investigate closure properties of DFA(2). We will refine this technique to prove the following theorem.

Theorem 2.1. *Let x be one of the following types of automata*: fa, ca, pda. *Then, for* $k>0$,

(a) $\mathrm{D}X(k)\cap B_2 \subsetneq \mathrm{D}X(k+1)\cap B_2$,

and, for $k>1$,

(b) $\quad DX(k)\cap B_2 \subsetneq NX(k)\cap B_2$.

For $k=1$ and $x=$ fa we have equality in (b). For $x=$ ca, pda it is not difficult to prove that $DX(1)\cap B_2 \subsetneq NX(1)\cap B_2$.

Thus, in particular, we obtain that $DPDA(k)\subsetneq DPDA(k+1)$ and $DPDA(k)\subsetneq NPDA(k)$ for $k>0$, which solves some long open problems about multihead pushdown automata. We strengthen also the known hierarchy results from [27, 42] by proving that they even hold for 2-bounded languages. Can we use here 1-bounded languages? No. For suppose that $L\in NPDA(k)$ and $L\subseteq a^*$. Then the set $\{x\in\mathbb{N}: a^x\in L\}$ is semilinear, so it is just a finite union of arithmetic sequences. Thus, L is regular and all hierarchies collapse to regular sets. It means that, in a sense, Theorem 2.1 is 'optimal'. We will also prove the following theorem.

Theorem 2.2. *Let* $x=$ dfa, dca, dpda. *Then, for* $k>1$, $X(k)$ *is not closed under union, intersection, and concatenation.*

Theorem 2.2 also holds for $k=1$ and $x=$ dca, dpda. For $x=$ dpda Theorem 2.2 solves some open problems stated by Harrison and Ibarra [13]. For $x=$ dfa the above results were already proved by Hromkovič [19].

The reader interested in separation techniques is advised to see the survey in [32] and the introductory part of [3].

The rest of the paper is divided into six sections. In Section 3 we will present some intuitions concerning the proof of the Fundamental Lemma. In Section 4 some number-theoretic and geometric properties of grid points of the plane will be proved. In Section 5 a new device, called a k-head pushdown automaton with l-bounded pushdown (dpda(k, l), in short), will be defined and we prove the normal-form lemma for dpda(k, l)'s. In Section 6 the notion of the tree of events for a dpda(k, l) will be introduced and some properties of this tree investigated. We will also prove there the Fundamental Lemma 6.6. In Section 7 we will derive from it our results for dfa(k)'s and dca(k)'s. In Section 8 we will prove that for 2-bounded inputs each dpda(k) can be substituted by an equivalent dpda(k, $2k$) and from this we will derive our results for dpda(k)'s.

3. Intuitions

Let $L_n=\{1^x2^y: y=ix$ for some integer $1\leqslant i\leqslant n\}$. We will use the L_n's as our witness languages. The most important and most difficult part of our proof is the Fundamental Lemma which in essence states that, for fixed k, l, there is an m such that $L_m\notin DPDA(k, l)$. Actually, we will prove that only a finite number of languages

L_n can be recognized by dpda(k, l)'s for fixed k and l. To understand the idea of the proof it is sufficient to consider only dfa(k)'s.

So, let A be a dfa(k) with state set Q and suppose that, in some configuration K of A on 1^x2^y for each $i = 1, 2, \ldots, k$, the distance between the position of h_i and the end of the block of letters scanned by h_i is greater than s, the number of states of A. Then, after several steps, the state of A must be repeated and A must enter a cycle. Let v_i be the number of times h_i is moved forward in this cycle. Then we may think of v_i as of the 'speed' of h_i. According to this intuition, each h_i moves with constant speed v_i until one of the heads reaches the end of the scanned block of letters. Also, the actual values of the v_i's are not essential. We can, for example, multiply each v_i by an integer.

To see this better, consider the following dfa(2) A recognizing L_1. A first places h_2 on the first symbol 2 and then enters a cycle in which both heads move one cell forward and A does not change its state. If both heads reach the ends of the scanned blocks of letters simultaneously, then A 'knows' that $y = x$. Similarly, moving h_2 m times faster A can check whether $y = mx$. Note now that in order to check whether $y = x$, A may have entered a cycle of any length, say m. What is really important is that the speed of h_1 is the same as the speed of h_2. However, if the cycle is $q_1, q_2, \ldots, q_m$, then A can compute $x \bmod m$ (or $y \bmod m$) by checking in which q_i h_1 (or h_2) reached the end of the block. In general, a dfa(k) has the ability to

(1) check linear equalities and inequalities between x and y with integer coefficients;

(2) check linear congruences involving x and y.

The reader probably feels that ability (2) does not help when the languages L_n are considered. Thus we will present a general idea of the proof by considering first, very informally, a simplified form of a dfa(k) which we will call a continuous-input k-head finite automaton, cfa(k) for short.

A cfa(k) A consists of a finite number of states. One of the states, say q_0, is initial, and F is the set of final states. The input to A is a pair of consecutive intervals of the real line $[0, x)$ and $[x, x+y)$, for some reals x, y. We denote such inputs by $w = (x, y)$. The first interval is painted with colour 1, the second interval is of colour 2, and the rest of the line is of colour 3. To each state an integer vector $v(q) = (v_1(q), \ldots, v_k(q))$ is assigned, where v_i is the speed of h_i in state q. A can change its state only at boundary configurations, that is, at such configurations when some head reaches the end of the scanned interval. A starts the computation in state q_0. The first head which will reach the end of the first interval will be h_d, where d is such that $v_d(q_0)$ is maximal (there can be several heads with the same speed). The new positions of the heads will be $xv_i(q_0)/v_d(q_0)$, $i = 1, \ldots, k$. Consider inputs (x, y) which have the same sequence of boundary configurations $q_0, q_1, \ldots, q_m, \ldots$. Then the position of the ith head, $i = 1, \ldots, k$, at the boundary configuration corresponding to q_m is a linear function $f_i(x, y)$. It can be easily proved by induction. For suppose that this is true for $q_0, \ldots, q_m$. Let h_d be the head such that if h_d reaches the end of the scanned interval first, then A enters q_{m+1}. (If q_{m+1} is entered after two heads

reach the ends of the intervals, then the proof is similar.) Let also $c_d = (x - f_d(x, y))/v_d(q_m)$ if $f_d(x, y) \leq x$, or $(x + y - f_d(x, y))/v_d(q_m)$ otherwise. Then the new positions at q_{m+1} are $f_i(x, y) + c_d v_i(q_m)$ otherwise. Then the new positions at q_{m+1} are $f_i(x, y) + c_d v_i(q_m)$, $i = 1, \ldots, k$, so they are linearly dependent on x and y, too.

Furthermore, we claim that the set of inputs (x, y) which have the same sequence of states at the boundary configurations is an angle of the form:

(*) $ax \leq y \leq bx$, for some rationals a, b (for $a > b$ it is an empty set).

For suppose that this is true for the sequence of states $q_0, q_1, \ldots, q_m$, and let X be this angle. Suppose that A enters q_{m+1} when h_d reaches the end of the scanned interval first. Then A will enter q_{m+1} after q_m iff, for each $i \neq d$, $c_d < c_i$. By the form of the c_i's these inequalities define an angle Y of the form (*). The appropriate set is the intersection of X and Y, which is again a set of the form (*). If q_{m+1} is entered with two heads, say h_p and h_r, entering a new interval, then we obtain the condition $c_p = c_r$, which, after intersection with X is also of the form (*). By induction, this proves the claim.

Observe now that the angle X at the boundary configuration after $q_0, q_1, \ldots, q_m$ is divided, when passing to another boundary configuration, into at most k smaller angles (depending on which head reaches a new boundary configuration) and into at most $k-1$ lines, corresponding to boundary configurations caused by two heads reaching the ends of the scanned intervals together. But the number of boundary configurations is at most $2k$. Therefore, the initial angle $\{(x, y) : x, y \geq 0\}$ is divided into at most $(2k-1)^{2k}$ angles, each corresponding to a final configuration of A. Suppose that A recognizes L_n with $n > (2k-1)^{2k}$. Then there will be an angle X corresponding to a final configuration K of A such that X contains two lines $y = ix$ and $y = jx$ for $1 \leq i < j \leq n$. Since A recognizes L_n, K must be accepting. But then A would also accept all points in X—a contradiction.

The general idea of the proof for dfa(k)'s is similar. We only have to appropriately redefine the notion of an angle. In the sequel it will be called a semisector (see next section). The main difficulty will be to show that the main properties of angles will be preserved (for example, the closure under intersection).

4. Seminets and semisectors

By $\mathbb{N}$ we denote the set of natural numbers and by $\mathbb{Z}$ the set of integers. 0 denotes the vector (0, 0). We also adopt the convention that

- other capital letters denote subsets of $\mathbb{N}^2$,
- t, u, v, w denote vectors, elements of $\mathbb{N}^2$,
- k, l denote arbitrary integers,
- other small letters, if not explicitly defined or if their meaning does not follow from the context, denote natural numbers.

In this section we will investigate properties of certain subsets of $\mathbb{N}^2$ called seminets and semisectors. Both seminets and semisectors are semilinear sets of special form.

P is a *net* if there exist a u_0 and a finite set $U=\{u_1, \ldots, u_n\}$ containing at least two linearly independent vectors such that

$$P = P(u_0; U) = \{u \in \mathbb{N}^2 : \text{there are } k_1, \ldots k_n \in \mathbb{Z} \text{ such that } u = u_0 + k_1 u_1 + k_2 u_2 + \cdots + k_n u_n\}.$$

Note the difference between this definition and the definition of a linear set: here, the numbers k_i need not be nonnegative. Intuitively, a net is a set of grid points in the plane forming a regular pattern. For example, the set P of vectors with even coordinates is a net because $P = P(0; (2,0), (0,2))$.

P is a *seminet* if either P is empty or is a finite union of nets. P is a 0-*seminet* if $0 \in P$.

We assume that $\mathbb{N}^2$ is partially ordered:

$$(x_1, y_1) \leqslant (x_2, y_2) \text{ iff } x_1 \leqslant x_2 \text{ and } y_1 \leqslant y_2.$$

For $i > 0$ we define

$$C_i = \{(x, y) : y = ix\}, \qquad D_n = \bigcup_{i \leqslant n} C_i, \qquad D = \bigcup_{i \in \mathbb{N}} C_i.$$

We also define $X - u = \{v : v + u \in X\}$.

Before stating the first lemma we encourage the reader to verify the following properties:

(P1) if $u \in P(u_0; U)$, then $P(u_0, U) = P(u; U)$;
(P2) if $u, v \in P(u_0; U)$ and $u \geqslant v$, then $u - v \in P(0; U)$;
(P3) if $u \in P(u_0; U)$, then $P(u_0; U) - u = P(0; U)$;
(P4) if $u \in P(u_0; U)$, then $u + v \in P(u_0 + v; U)$.

Lemma 4.1. *If P and R are seminets, then so is $P \cap R$.*

Proof. We can assume that P and R are nets, that is, $P = P(u_0; U)$ and $R = P(v_0; V)$. Suppose that $P \cap R \neq \emptyset$ and let S be the set of all minimal elements in $P \cap R$, and T the set of all minimal elements in $Q - \{0\}$, where $Q = P(0; U) \cap P(0; V)$. Then S and T are finite (see, for example, [8, Corollary 5.4.1, p. 164]).

We first show that T contains two linearly independent vectors. U contains a base of the 2-dimensional real linear space, so $(1, 0)$ is a linear combination of vectors in U with rational coefficients. Then $(x_1, 0) \in P(0; U)$ for some x_1. Similarly, $(x_2, 0) \in P(0; V)$ for some x_2. Therefore, $(x, 0) \in Q$, where $x > 0$ is any common multiple of x_1, x_2. The same argument gives us that $(0, y) \in Q$ for some $y > 0$. The only vector which is smaller than $(x, 0)$ and $(0, y)$ is 0, so T must contain two linearly independent vectors.

We now prove that $Q = P(0; T)$. The inclusion $\supseteq$ is obvious. To prove the other one, suppose that there is $v \in Q - P(0; T)$ and take such a minimal v. Then $v > t$ for some $t \in T$. This easily implies that $v - t \in Q - P(0; T)$ which gives a contradiction because $v - t < v$.

Finally, we prove that $P \cap R = \bigcup_{u \in S} P(u; T)$. Take any $u \in S$. Since $u \in P$ and $T \subseteq P(0; U)$, we have $P(u; T) \subseteq P$. Similarly, $P(u; T) \subseteq R$. This proves the $\supseteq$-inclusion.

To prove the other inclusion, observe that if $v \in P \cap R$, then $v \geqslant u$ for some $u \in S$. Since $u, v \in P$ and $u, v \in R$, we have $v - u \in P(0; U)$ and $v - u \in P(0; V)$ which together imply that $v - u \in Q$. But then $v \in P(u; T)$ because, as it was shown, $Q = P(0; T)$. This proves the $\subseteq$-inclusion. □

Lemma 4.2. *Let* $P = \{(x, y) : ax + by \equiv c \pmod{d}\}$, *where* $a, b, c \in \mathbb{Z}$ *and* $d \in \mathbb{N}^+$. *Then* P *is a seminet.*

Proof. Let S be the set of minimal elements in P, and T the set of minimal elements in the set $\{(x, y) : (x, y) \neq 0$ and $ax + by \equiv 0 \pmod{d}\}$. Then, similarly as in Lemma 4.1, $P = \bigcup_{u \in S} P(u; T)$. □

If $a, b, d, e \in \mathbb{N}$ are such that $a + b > 0$, $d + e > 0$ and $c, f \in \mathbb{Z}$, then the set

$$S = S(a, b, c, d, e, f) = \{(x, y) : -ax + by + c > 0 \text{ and } -dx + ey + f < 0\}$$

is called a *sector*. Thus, S is the set of grid points in the plane with nonnegative integer coordinates lying between the lines $-ax + by + c = 0$ and $-dx + ey + f = 0$.

If $X = S \cap P - F$, for a sector S, a seminet P, and a finite set F, then X is called a *semisector*. If, moreover, $P \neq \emptyset$ and $ae < bd$, then X is said to be a *proper semisector*. An example of a proper semisector is shown in Fig. 1.

Let X be finite and z such that $(z - 1, z - 1) \geqslant (x, y)$ for every $(x, y) \in X$. Then,

$$X \subseteq S = S(1, 0, z, 0, 1, -z) = \{(x, y) : (x, y) \leqslant (z - 1, z - 1)\}.$$

Therefore, $X = S - (S - X)$ is a semisector.

It is easy to see that a nonproper semisector is either finite or is a set of grid points between two parallel lines.

If $X_i = S(a, b, c_i, d, e, f_i) \cap P_i - F_i$, where P_i is a seminet and F_i is finite, for $i = 1, 2$, then the semisectors X_1, X_2 are called *parallel*. We immediately obtain that all finite sets are pairwise parallel.

Two semisectors X_1, X_2 are *independent* if there are S_i, P_i, F_i such that $X_i = S_i \cap P_i - F_i$, for $i = 1, 2$, and $S_1 \cap S_2$ is not a proper semisector (we will prove later that it is a semisector).

Lemma 4.3. *Let* $X = \{(x, y) : -px + qy + r > 0\}$, *where* $p, q \in \mathbb{N}$, $r \in \mathbb{Z}$ *and* $-p + q > 0$. *If* S *is a sector, then* $X \cap S$ *is a semisector. Moreover, if* $X' = \{(x, y) : -px + qy + s > 0\}$, *then* $X \cap S$ *and* $X' \cap S$ *are parallel semisectors.*

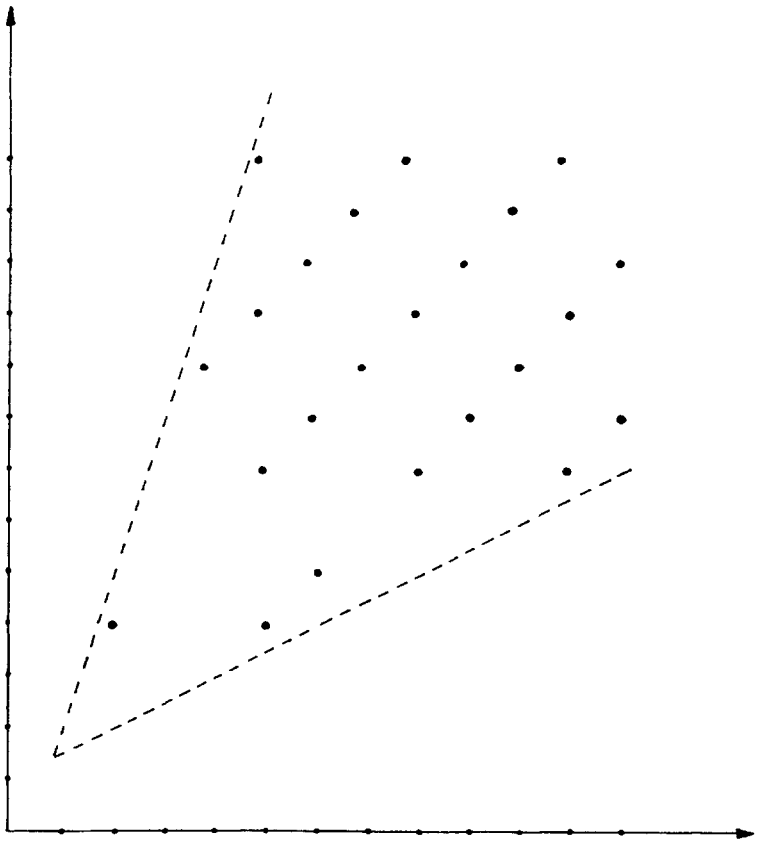

Fig. 1. $P = \{(x, y) : x + 2y = 1 \pmod 3\}$
$= P((1, 0);\ (3, 0), (0, 3)) \cup P((2, 1);\ (\bar{3}, 0), (0, 3)) \cup P((3, 2);\ (3, 0), (0, 3))$,
$X = S(1, 2, -2, 3, 2, 2)$, the area between the broken lines,
$F = \{(3, 5), (4, 6), (7, 6)\}$,
$Y = X \cap P - F$, the set of dotted points.

Proof. Let $S = S(a, b, c, d, e, f)$. We assume that S is proper, that is, $ae < bd$. The case when S is not proper is left to the reader. We consider several cases.

Case 1: $pe > qd$. Then $X \cap S$ is finite.

Case 2: $pe \leqslant qd$ and $aq < bp$. Then $X \cap S = S(p, q, r, d, e, f) - F$, where

$$F = \{(x, y) : -ax + by + c \leqslant 0 \text{ and } -px + qy + r > 0 \text{ and } -dx + ey + f < 0\}$$

is finite.

Case 3: $aq = bp$. Then we can assume that $a = p$ and $b = q$. Then $X \cap S = S(a, b, \min(c, r), d, e, f)$.

Case 4: $aq > bp$. Then $F = S - X$ is finite and $X \cap S = S - F$ is a semisector.

Repeating this argument simultaneously for X and X' we obtain the second part of the lemma. □

Lemma 4.4. *If X_i, $i = 1, \ldots, n$, are semisectors, then $X = \bigcap_{i \leqslant n} X_i$ is a semisector. Moreover, if, for $i = 1, \ldots, n$, Y_i is a semisector parallel to X_i, then $Y = \bigcap_{i \leqslant n} Y_i$ is a semisector parallel to X.*

Proof. It is enough to consider the case $n=2$. Let $X_i = S_i \cap P_i - F_i$, where $S_i = S(a_i, b_i, c_i, d_i, e_i, f_i)$, P_i is a seminet, and F_i is finite, for $i=1,2$. Then $S_2 = U \cap V$, where

$$U=\{(x,y): -a_2x+b_2y+c_2>0\}, \qquad V=\{(x,y): -d_2x+e_2y+f_2<0\}.$$

So we obtain

$$\begin{aligned} X = X_1 \cap X_2 &= (S_1 \cap P_1 - F_1) \cap (U \cap V \cap P_2 - F_2) \\ &= (S_1 \cap U) \cap V \cap P_1 \cap P_2 - F_3, \end{aligned}$$

where $F_3 = F_1 \cup F_2$ is finite. Using Lemma 4.3 we get

$$X = (S \cap P - F) \cap V \cap P_1 \cap P_2 - F_3,$$

for some sector S, seminet P, and finite set F. Therefore,

$$X = S \cap V \cap P \cap P_1 \cap P_2 - F_4,$$

where $F_4 = F_3 \cup F$ is finite. Using again Lemma 4.3 we get

$$X = (S' \cap P' - F') \cap P \cap P_1 \cap P_2 - F_4,$$

for some sector S', seminet P', and finite set F'. Now we have

$$X = S' \cap (P' \cap P \cap P_1 \cap P_2) - F_5,$$

where $F_5 = F_4 \cup F'$ is finite. So X is a semisector because $P' \cap P \cap P_1 \cap P_2$ is a seminet from Lemma 4.1.

The second part of the proof is analogous to the first one. We only have to repeat the above argument simultaneously for X and Y, using now the second part of Lemma 4.3. □

Lemma 4.5. *Let $X_i = \{(x,y): ax+by+c_i>0\}$ and $Y_i = X_i \cap P_i - F_i$, where a, b, c_i are rational numbers, P_i is a seminet, and F_i is finite for $i=1,2$. Then Y_1, Y_2 are parallel semisectors.*

Proof. A straightforward verification. □

Lemma 4.6. *If X, Y are semisectors, if $X \subseteq Y$, and X is proper, then Y is proper too.*

Proof. Obvious. □

Lemma 4.7. *Let P_i, $i=1,\ldots,n$ be nets. Then there are $g, h>0$ such that, for every $i=1,\ldots,n$, $P_i-(g,0)=P_i-(0,h)=P_i$.*

Proof. Suppose $P(u; U)$ is a net, and $P(0; U)-v=P(0; U)$. Then, obviously, $v \in P(0; U)$ and consequently, $P(u; U)-v=P(u; U)$. So it is enough to prove the lemma for $P_i = P(0; U_i)$, $i=1,\ldots,n$. The existence of g, h can now be established by an argument analogous to the part of the proof of Lemma 4.1 where we showed that the set T contains two linearly independent vectors. □

Lemma 4.8. *Let S be a proper sector and $g, h > 0$. Then there are numbers $p, q > 0$ such that $\bigcup_{i \in \mathbb{N}} (S - (ipg, iqh)) = \mathbb{N}^2$.*

Proof. Let $S = S(a, b, c, d, e, f)$. From the definition of a proper sector, $ae < bd$ and $b, d > 0$. We will first find $p, q > 0$ such that

(*) $apg > bqh$ and $qhe < pgd$

(that is, the line $bqx = pgy$ is between the lines $ax = by$, $dx = ey$).

If $a = 0$, we take $q = 1$ and $p > qheg^{-1}d^{-1}$. Suppose now $a > 0$. We must find p, q such that $ehd^{-1}g^{-1} < pq^{-1} < bha^{-1}g^{-1}$. The existence of such p, q is now obvious because $ae < bd$.

To prove the lemma it suffices to show that

(**) for each $(x, y) \in \mathbb{N}^2$ there is an $i \in \mathbb{N}$ such that $(x, y) + (ipg, iqh) \in S$.

Consider the inequality $-a(x + ipg) + b(y + iqh) + c > 0$. This is equivalent to $-ax + by + c + i(bqh - apg) > 0$, which is true for i large enough because of (*). Similarly, $-d(x + ipg) + e(y + iqh) + f < 0$ for i large enough. This proves (**). □

Lemma 4.9. *Let P, R be seminets and S a proper sector. Then $P \cap S = R \cap S$ implies $P = R$.*

Proof. If $P \cap S = R \cap S$ then, for any u, $(P \cap S) - u = (R \cap S) - u$, that is, $(P - u) \cap (S - u) = (R - u) \cap (S - u)$. Let P_i, $i = 1, \ldots, n$, be the nets in the descriptions of P and R, and take g, h satisfying Lemma 4.7. Then, from Lemma 4.7, for any i, j,

$$P - (ig, jh) = P \quad \text{and} \quad R - (ig, jh) = R.$$

Let p, q be the numbers from Lemma 4.8 and $X_i = S - (ipg, iqh)$ for $i \geqslant 0$. Then, for any i,

$$(P - (ipg, iqh)) \cap X_i = (R - (ipg, iqh)) \cap X_i,$$

which implies $P \cap X_i = R \cap X_i$. Thus, $\bigcup_{i \in \mathbb{N}} (P \cap X_i) = \bigcup_{i \in \mathbb{N}} (R \cap X_i)$, that is, $P \cap \bigcup_{i \in \mathbb{N}} X_i = R \cap \bigcup_{i \in \mathbb{N}} X_i$, and, from Lemma 4.8, $P = R$. □

Lemma 4.10. *Let $X = S \cap P - F$ be a semisector, where S is a sector, P is a seminet and F is finite. Suppose that*

$$X = \bigcup_{i \leqslant n} X_i \cup \bigcup_{j \leqslant m} Y_j,$$

where

(a) *$X_1, \ldots, X_n, Y_1, \ldots, Y_m$ are semisectors,*

(b) *$X_1, \ldots, X_n$ are parallel and proper,*

(c) *for $i = 1, \ldots, n$, $j = 1, \ldots, m$, X_i and Y_j are independent.*

Let $X_i = S_i \cap P_i - F_i$, where S_i is a sector, P_i is a seminet, and F_i is finite, for $i = 1, \ldots, n$. Then $P = \bigcup_{i \leqslant n} P_i$.

Proof. Let $S_i = S(a, b, c_i, d, e, f_i)$, $i = 1, \ldots, n$. Take c small enough and f large enough such that for $S_0 = S(a, b, c, d, e, f)$

$$(*) \qquad S_0 \subseteq S \cap \bigcup_{i \leq n} S_i - \left(\bigcup_{i \leq n} F_i \cup F \cup \bigcup_{j \leq m} Y_j \right).$$

The existence of such numbers c, f can be shown by considering appropriate linear inequalities. Then $X \cap S_0 = (S \cap P - F) \cap S_0 = P \cap S_0$. On the other hand,

$$\begin{aligned} X \cap S_0 &= \left(\bigcup_{i \leq n} X_i \cup \bigcup_{j \leq m} Y_j \right) \cap S_0 \\ &= \bigcup_{i \leq n} (X_i \cap S_0) = \bigcup_{i \leq n} (S_i \cap P_i - F_i) \cap S_0 \\ &= \bigcup_{i \leq n} (S_0 \cap P_i) = S_0 \cap \left(\bigcup_{i \leq n} P_i \right). \end{aligned}$$

From Lemma 4.9 we obtain that $P = \bigcup_{i \leq n} P_i$. □

Lemma 4.11. *If P is a 0-seminet, then, for every $i > 0$, $P \cap C_i$ is infinite.*

Proof. Let $P(u_0; U) \subseteq P$ be a net containing 0, and $i > 0$ be fixed. We can take $u_0 = 0$. Let $u_1, u_2 \in U$ be linearly independent and $R = P(0; u_1, u_2) \cap C_i$. If $v \in R$, then $jv \in R$ for any j. Therefore it is enough to prove that R is nonempty. Consider the equation $a_1 u_1 + a_2 u_2 = (x, ix)$. From the linear independence of u_1, u_2, the above equation has a solution a_1, a_2 in rational numbers. For some j, $k_1 = ja_1$ and $k_2 = ja_2$ are integers. Then $k_1 u_1 + k_2 u_2 = (jx, ijx) \in R$. □

Lemma 4.11 is an important property of 0-seminets. It does not hold in general for all nets. Consider the following example. Let $P = P_1 \cup P_2$, where

$$P_1 = \{(x, y) : x + y \equiv 2 \pmod 6\}, \qquad P_2 = \{(x, y) : x + y \equiv 3 \pmod 6\}.$$

Then $C_1 \cap P$ and $C_2 \cap P$ are infinite, but $C_1 \cap P_2 = \emptyset$ and $C_2 \cap P_1 = \emptyset$. We define now languages

$$M_1 = \{1^x 2^y : (x, y) \in C_1 \cap P\}, \qquad M_2 = (1^x 2^y : (x, y) \in C_2 \cap P\}$$

and $M = M_1 \cup M_2$. Then the languages M_1, M_2 can be separated using states only. More formally, a dfa(1) having on the input only words from M can tell whether an input comes from M_1 or M_2. It has only to count $(x + y)$ mod 6. Such a separation is not possible if P contains 0, because, in this case, some P_i also contains 0 and $C_1 \cap P_i$ and $C_2 \cap P_i$ are both infinite from Lemma 4.11.

Lemma 4.12. *If X is a proper semisector, then $X - D$ is infinite.*

Proof. We can assume that $X = S \cap P$, where S is a sector and P is a net, $P = P(u; U)$. Let $S = S(a, b, c, d, e, f)$ and take i to be smallest number such that $ib > a$. Then $Y = S(a, b, c, i, 1, 0)$ is a proper sector such that $Y \cap D = \emptyset$ and $Y \subseteq X$. The lemma now easily follows from the fact that $Y \cap P$ is infinite (the proof of this is left to the reader). □

Lemma 4.13. *Let X be a semisector such that $X \cap C_i$ and $X \cap C_j$ are infinite for $i > j > 0$. Then X is proper.*

Proof. A simple verification. □

5. Pushdown automata with bounded stack

We will consider only automata accepting 2-bounded languages. Without any loss of generality we can assume that
- every input is of the form 1^x2^y, where $x, y > 0$;
- the endmarker 3 occurs at the end of the input tape.

Let A be a dpda(k, l) with a set of states Q such that $L(A) \subseteq 1^+2^+$. Let push(a_i) and pop denote the operations of A on the stack. We will abbreviate push(a_i) by push$_i$. From the definition of a dpda$(k, 1)$ the contents of the stack is always of the form

$$a_1^{r_1} a_2^{r_2} \dots a_l^{r_l}$$

We will usually refer to $a_i^{r_i}$ as to *the ith segment of the stack.*

A *configuration* of A on 1^x2^y is a $(k+l+1)$-tuple $K = (q, p_1, \dots, p_{k+l})$, where $q \in Q$, $p_1, \dots, p_k \in \{1, \dots, x+y+1\}$ are the positions of the heads of A, and $p_{k+1}, \dots, p_{k+l} \in \mathbb{N}$ are the lengths of the segments of the stack of A in K.

If K is as above, then we write $q(K) = q$ and $p_t(K) = p_t$ for each t, and by $\text{top}_A(K)$ (or simply $\text{top}(K)$ if A is understood) we denote the maximum $t > k$ such that $p_t \neq 0$, or 0 if such t does not exist. Thus, by the definition of a dpda(k, l), A cannot execute push$_i$ in K when $\text{top}(K) > i$.

$K_0 = (q_0, 1, \dots, 1, 0, \dots, 0)$ is the initial configuration of A. The computation of A is defined as usual (see, for example, [13]).

If K is a configuration of A, then by the *mode* of A in K we understand a $(k+2)$-tuple $M = M(K) = (q, e_1, \dots, e_k, i)$, where $q \in Q$, for every t, $e_t \in \{1, 2, 3\}$ is the symbol scanned by the tth head, and $i = \text{top}(K)$. If M is as above, then we define $q(M) = q$ and, for every $t \leq k$, $e_t(M) = e_t$, and $\text{top}(M) = i$.

Now we define events in the computation of A. Events are those configurations in which one of the heads reaches the end of the block of letters on the input, or the top symbol on the stack changes. We do not care about changes of states.

Formally, a configuration K is an *event* in the computation of A on 1^x2^y if either $K = K_0$ (in this case, K is said to be of type (ev0)), or if K' is the configuration of A occurring immediately before K, then one of the following cases occurs:

(ev1) $e_t(M') < 3$ and $e_t(M) = e_t(M') + 1$ for some $t \leq k$;

(ev2) $\text{top}(M') > 0$ and $\text{top}(M) < \text{top}(M')$;

(ev3) $\text{top}(M') < l$ and $\text{top}(M) > \text{top}(M')$,

where $M = M(K)$ and $M' = M(K')$.

Suppose now that $\text{top}(K) = i < l$ and, being in K, a dpda(k, l) A begins to execute the following operations: push_{i+1}, pop, push_{i+1}, pop, ... Then an event will occur in every step. But in some sense these events are inessential; they can be easily eliminated by using more states. This observation leads to the definition of the normal form of a dpda(k, l).

A dpda(k, l) is said to be in *normal form* if in every computation the following conditions are satisfied:

(nf1) A never moves two heads, or moves a head and changes the stack simultaneously;

(nf2) A never stops before all the heads reach the endmarker and the stack is empty, regardless from which configuration it started (this technical requirement will simplify some definitions in Section 6);

(nf3) A never enters a loop (that is, A is never in the same configuration twice);

(nf4) A does not increase the stack when all the heads are on the endmarker;

(nf5) A does not not make reversals on the stack between any two events of type (ev0)-(ev2). More formally, if $K_1, \ldots, K_n$ is a sequence of consecutive configurations of A, and K_1 and K_n are the only events (ev0)-(ev2) in this sequence, then there are no $1 \leq i, j < n$ such that A makes a pop in K_i and a push in K_j; the reversals may only happen in K_1 or K_n;

(nf6) each event (ev3) immediately occurs after an event of type (ev0)-(ev2). Furthermore, if K is an event of type (ev0)-(ev2) after which an event K' of type (ev3) occurs, then $q(K) = q(K')$, that is, A does not change its state.

Lemma 5.1. *If a* dpda(k, l) A *is in normal form, then in every computation of* A *the number of events* (ev0)-(ev2) *is at most* $4k+1$, *and the number of events* (ev3) *is at most* $2k$.

Proof. Suppose that an event (ev3) occurred. This means that A made push_t for some t. But, from (nf5) A cannot make a pop until an event (ev1) occurs. There is exactly one event (ev0) and $2k$ events (ev1) in every computation. It implies that the number of events (ev3) is at most $2k$. Therefore, the number of events (ev2) is also at most $2k$. This gives in total at most $4k+1$ events (ev0)-(ev2). □

Note that the bound in Lemma 5.1 does not depend on l.

Lemma 5.2. *For every* dpda(k, l) A *there exists a* dpda(k, l) B *satisfying* (nf1)-(nf4) *such that* $L(B) = L(A)$.

Proof. By standard methods of automata theory. □

Lemma 5.3. *For every* dpda(k, l) A *there is a* dpda(k, l) B *in normal form such that* $L(B) = L(A)$.

Proof. By Lemma 5.2 we can assume that A satisfies (nf1)-(nf4). Let A_i, $0 \leq i \leq 2k$, be a dpda(k, l) equivalent to A satisfying (nf1)-(nf4) and, additionally, satisfying (nf5) and (nf6) between (ev0) and the ith event (ev1). We can take A_0 to be A. Since A_{2k} is in normal form, we can take B to be A_{2k}. Therefore, it remains to show that we can construct A_{i+1} from A_i for $i = 0, 1, \ldots, 2k-1$. Let $0 \leq i \leq 2k-1$ and let n be the number of states of A_i. We first construct a dpda(k, l) A_i' which is the same as A_i except that it has a bounded-length buffer BUFF in the finite memory used to store small portions of the stack of A_i.

If A_i has $a_1^{r_1} \ldots a_l^{r_l}$ on the stack, then A_i' stores $a_1^{n_1} \ldots a_l^{n_l}$, for $n_j = \min(r_j, n)$, $j = 1, \ldots, l$ in BUFF and $a_1^{r_1 - n_1} \ldots a_l^{r_l - n_l}$ on the stack.

So A_i' has additionally the following information. It 'knows' for every t whether the tth segment of the stack of A_i has length at least n because then BUFF contains a_t^n. If this is not the case, this segment is completely stored in BUFF. A straightforward construction of A_i' is left to the reader.

We will construct now A_i'' using BUFF and other buffers to store small changes of the stack. These buffers amortize sudden changes of the behaviour of A_i between the ith and $(i+1)$st events (ev1). A_i'' makes exactly the same moves as A_i' until the ith event K of type (ev1) occurs. Now, let $t = \mathrm{top}_{A_i}(K)$. A_i'' will use buffers BUFF, $\mathrm{BUFF}_t, \ldots, \mathrm{BUFF}_l$, where BUFF is the same as in A_i' and, for each $p = t, \ldots, l$, BUFF_p is used to store words a_p^m, where $m \leq n$. The whole contents of the stack is stored as

$$a_1^{n_1 + r_1} \ldots a_{t-1}^{n_{t-1} + r_{t-1}} a_t^{n_t + r_t + m_t} a_{t+1}^{m_{t+1}} \ldots a_l^{m_l},$$

where $\mathrm{BUFF} = a_1^{n_1} \ldots a_t^{n_t}$, $a_1^{r_1} \ldots a_t^{r_t}$ is stored on the stack of A_i'' and $\mathrm{BUFF}_p = a_p^{m_p}$, $p = t, \ldots, l$.

Starting from K, A_i'' uses the buffers to simulate the operations of A_i on the stack as long as possible, that is, until one of the following occurs:

(a) $m_p = n$ for some p and A_i makes push_p (BUFF_p overflows),

(b) $n_t + m_t = l$, $m_p = 0$ for $p = t+1, \ldots, l$ and A_i makes a pop.

Case (a): In this case A_i has entered a cycle in which it will increase the pth segment of the stack. Note that in this cycle A_i may make also some pop's and operations push_r for $r > p$. To smooth this behaviour of A_i, A_i'' uses again the buffers $\mathrm{BUFF}_p, \ldots, \mathrm{BUFF}_l$. Each operation other than push_p is performed on these buffers. Also, if A_i makes push_p and BUFF_p is not full (it stores a_p^m for $m < n$), then A_i''

uses BUFF_p instead of its stack. Since A_i has entered the cycle, BUFF_p will never be emptied. Also, A_i'' does not make reversals because it makes only one operation on the stack: push_p.

Case (b): We have two subcases here.

Case (b1): $r_t = 0$. It means that the tth segment of the stack of A_i would be empty now. This is an event K' of A_i of type (ev2) A_i'' repeats the whole procedure again with $t' = \mathrm{top}_{A_i}(K')$ instead of t.

Case (b2): $r_t > 0$. Therefore, initially, BUFF contained a subword a_t^n. Similarly as in Case (a), this implies that A_i must have entered a cycle, but now the total change of the tth segment of the stack in this case is negative. The operations of A_i'' are now symmetric to those in Case (a). While the top symbol on the stack of A_i is a_p, for $p \geqslant t$, and no event (ev1) occurs, A_i'' proceeds as follows. If A_i makes push_p, for some $t \leqslant p \leqslant l$, or if A_i makes a pop and the top symbol is not a_t, then A_i'' appropriately changes BUFF_p. If A_i makes a pop and the top symbol is a_t, then A_i'' looks at the contents of BUFF_t. If BUFF_t is empty, then A_i'' also makes a pop on the stack, otherwise it makes a pop on BUFF_t.

These actions are repeated until the tth segment of the stack of A_i (or, equivalently, of A_i'') is empty. Similarly as in Case (b1), A_i'' repeats the whole procedure.

The simulation above is stopped by the next event (ev1). Then A_i'' behaves exactly as A_i' remembering that some of the stack contents may be still stored in the buffers $\mathrm{BUFF}_t, \ldots, \mathrm{BUFF}_l$.

A_i'' satisfies (nf1)-(nf5) before the $(i+1)$st event (ev1). To construct A_{i+1} we have to modify it to satisfy also (nf6). This can be done using technical tricks similar to those in the above proof. □

6. The tree of events

For a dpda(k, l) A in normal form we define the tree $\mathrm{ET}(A)$, called *the tree of events* of A, which will represent, in a sense, all possible computations of A. The nodes of $\mathrm{ET}(A)$ are labelled by modes of A. If E is a node, then $\mathrm{lab}(E)$ is the label of E. Moreover,

(et0): E_0 with $\mathrm{lab}(E_0) = M_0$ is the root of $\mathrm{ET}(A)$.

Let E be a node with $\mathrm{lab}(E) = (q, e_1, \ldots, e_k, i)$. If in the mode $M = (q, e_1, \ldots, e_k, i)$ A does not make push_{i+1}, then, for $p \in Q$ and $1 \leqslant t \leqslant k$,

(et1): E has a son F with $\mathrm{lab}(F) = (p, e_1, \ldots, e_t + 1, \ldots, e_k, i)$ if $e_t < 3$;

(et2): E has a son G with $\mathrm{lab}(G) = (p, e_1, \ldots, e_k, i-1)$ if $i > 0$.

Otherwise:

(et3): E has a single son H with $\mathrm{lab}(H) = (q, e_1, \ldots, e_k, i+1)$.

Furthermore, if case (et3) occurs, then E is called *degenerate.*

Note that some labels may appear several times in $\mathrm{ET}(A)$.

Remark 6.1. As it was pointed out by some referees, it would be more natural to

consider here a dag in place of a tree. However, since we are rather interested in paths (corresponding to sequences of boundary configurations) than in vertices, it is technically easier to talk about a tree since then each vertex uniquely determines the path joining it with the root.

Instead of $q(\mathrm{lab}(E))$ we will write shortly $q(E)$. Immediately from the definition and Lemma 5.1 we obtain the two following facts.

Fact 6.2. *If $K_0, \ldots, K_n$ is the sequence of events in some computation of a* dpda(k, l) *A, then there is a path $E_0, \ldots, E_n$ in* ET(A) *such that* $\mathrm{lab}(E_i) = M(K_i)$ *for* $i = 1, \ldots, n$.

Fact 6.3. *If a* dpda(k, l) *A is in normal form, then the depth of* ET(A) *is at most* $6k$. *For each path from E_0 to a leaf, the number of inner nondegenerate nodes* (et0)-(et2) *on this path is at most* $4k$.

Let now A be a dpda(k, l) in normal form. We define the functions

$$\mathrm{DOM}: \mathrm{ET}(A) \to 2^{\mathbb{N}^2}, \qquad p_t: \mathrm{ET}(A) \times \mathbb{N}^2 \to \mathbb{N} \quad \text{for } t = 1, \ldots, k+l,$$

as follows (ambiguously, ET(A) above denotes the set of nodes): Let E be a node in ET(A) and $E_0, \ldots, E_n = E$ the path from E_0 to E. If (x, y) is such that the sequence of the first n events of A on 1^x2^y is $K_0, \ldots, K_n$ and, for $i = 0, \ldots, n$, it holds that $M(K_i) = \mathrm{lab}(E_i)$, then $(x, y) \in \mathrm{DOM}(E)$ and $p_t(E, x, y) = p_t(K_n)$. Moreover, DOM(E) contains only pairs (x, y) satisfying the above condition.

So DOM(E) is the set of those (x, y)'s such that the computation of A on 1^x2^y has the sequence of events $\mathrm{lab}(E_0), \mathrm{lab}(E_1), \ldots, \mathrm{lab}(E_n), \ldots$. To explain the meaning of p_t, let E be fixed. Then DOM(E) is the domain of $p_t(E, *, *)$. For $(x, y) \in \mathrm{DOM}(E)$, $p_t(E, x, y)$ is either the position of the tth head or the length of the $(t-k+1)$st segment of the stack in K_n, depending on whether $t \leq k$ or $t > k$. Since A is deterministic, the functions p_t are well defined. Note that all these functions also depend on A but, for conciseness, we will consequently omit appropriate subscripts.

Let G be an inner nondegenerate node in ET(A) but not a leaf, and set $z = |Q| + 1$, where Q is the set of states of A. Consider a configuration K of A on 1^z2^z such that $q(K) = q(G)$, and

$$p_t(K) = \begin{cases} (e_t(\mathrm{lab}(G)) - 1)z + 1 & \text{for } t \leq k, \\ z & \text{for } t = \mathrm{top}(\mathrm{lab}(G)) > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Note that A may never enter K when computing on 1^z2^z. Consider, however, the computation of A starting from K: $K = K_1, \ldots, K_z$. Such a computation exists because A is in normal form. Let $1 \leq m < n \leq z$ be the smallest numbers such that $M(K_m) = M(K_n)$ (we can only require that $q(K_m) = q(K_n)$). Then we define for each t:

$$l_t(G) = p_t(K_n) - p_t(K_m),$$

$$g_t(G, q(K_i)) = p_t(K_i) - p_t(K_1) \quad \text{for } i = 1, \ldots, n-1.$$

Intuitively, $K_m, \ldots, K_{n-1}$ is a cycle and $l_t(G)$ is the total change of
- the position of the tth head for $t \leq k$,
- the length of the $(t-k+1)$st segment of the stack for $t > k$,

during the cycle.

The meaning of $g_t(G, q)$ is similar: the words 'during the cycle' should be substituted by 'before A enters state q'. We also define two sets:

$$\mathrm{SP}(G) = \{q(K_1), \ldots, q(K_{m-1})\}, \qquad \mathrm{LP}(G) = \{q(K_m), \ldots, q(K_{n-1})\}.$$

Informally, by checking if $q \in \mathrm{SP}(G)$ or $q \in \mathrm{LP}(G)$ we can verify whether A, after the event corresponding to G and being in state q, has already entered a cycle or not.

Moreover, suppose that E, F are sons of G such that $\mathrm{lab}(E)$ and $\mathrm{lab}(F)$ differ only in the first coordinate and either $q(E), q(F) \in \mathrm{SP}(G)$ or $q(E), q(F) \in \mathrm{LP}(G)$. Then E, F are called *twins*.

Lemma 6.4. *Let A be a* dpda(k, l) *in normal form, and E a node in* ET(A). *Then*

(a) DOM(E) *is a semisector*;

(b) *if F is a twin of E, then* DOM(E) *and* DOM(F) *are parallel semisectors*;

(c) *if F is a brother of E but not a twin, then* DOM(E) *and* DOM(F) *are independent*;

(d) *for every t there exist natural numbers a_t, b_t, and c_t such that $p_t(E, x, y) = a_t x + b_t y + c_t$, for $(x, y) \in \mathrm{DOM}(E)$.*

Proof. The proof is by induction on the depth of E in ET(A). The case $E = E_0$ is trivial: $\mathrm{DOM}(E) = S(0, 1, 0, 1, 0, 0)$ and

$$p_t(E, x, y) = \begin{cases} 1 & \text{for } t \geq k, \\ 0 & \text{for } t > k. \end{cases}$$

Suppose now that $E \neq E_0$, and G is a father of E. We can assume that G is nondegenerate. Let F be a twin of E. For any son H of G we define the number change(H), $1 \leq \mathrm{change}(H) \leq k + l$. Let $\mathrm{lab}(G) = (q, e_1, \ldots, e_k, i)$, and $\mathrm{lab}(H) = (p, f_1, \ldots, f_k, j)$. Then,

$$\mathrm{change}(H) = \begin{cases} t & \text{if } e_t \neq f_t \text{ for some } 1 \leq t \leq k, \\ k + i & \text{if } i \neq j. \end{cases}$$

Since A is in normal form, no two cases can occur simultaneously, so change(H) is well defined.

Assume that the lemma holds for G. We will show that it also holds for E and F. We have that DOM(G) is a semisector and $p_t(G, x, y) = a_t x + b_t y + c_t$ for each t.

For $t \leq k$ we define

$$d_t = \begin{cases} 0 & \text{if } e_t(G) = 1, \\ 1 & \text{if } e_t(G) = 2, 3. \end{cases}$$

Let us denote shortly $g_t(G, q)$ by $g_t(q)$ and $l_t(G)$ by l_t. Now, for each son H of G we define:

$$\text{dist}_t(H, x, y) = \begin{cases} x + d_t y + 1 - p_t(G, x, y) - g_t(q(H)) & \text{for } t \leq k, \\ p_t(G, x, y) + g_t(q(H)) & \text{for } t > k. \end{cases}$$

The intuition is as follows. Suppose that $(x, y) \in \text{DOM}(H)$ and let K_1, K_2 be the events corresponding to G and H in the computation of A on $1^x 2^y$. Let K_3 be the first configuration after K_1 such that $q(K_3) = q(K_2)$ $(= q(\text{lab}(H)))$. Then, for $t \leq k$, $\text{dist}_t(H, x, y) = x + d_t y + 1 - p_t(K_3)$ is the distance between the position of the tth head in K_3 and the first symbol different from the one scanned in K_3. For $t > k$, $\text{dist}_t(H, x, y) = p_t(K_3)$ is the length of the $(t-k)$th segment of the stack.

Suppose now that F is a twin of E and let $s = \text{change}(F) = \text{change}(E)$. We first prove parts (a), (b), (d) by considering two cases.

Case 1: $q(E)$, $q(F) \in \text{SP}(G)$. For $H = E, F$, we define $B_s(H) = \{(x, y): \text{dist}_s(H, x, y) = 0\}$ and, for $t \neq s$, $B_t(H) = \{(x, y): \text{dist}_t(H, x, y) > 0\}$.

Then, by Lemma 4.5, $B_t(E)$ and $B_t(F)$ are parallel semisectors for each t. But

$$\text{DOM}(H) = \text{DOM}(G) \cap \bigcap_{t=1}^{k+l} B_t(H).$$

Therefore, using Lemma 4.4, we obtain parts (a) and (b). Part (d) is immediate because, for each t,

$$p_t(E, x, y) = p_t(G, x, y) + g_t(q(E)).$$

Case 2: $q(E), q(F) \in \text{LP}(G)$. For $H = E, F$ let

$$B_s(H) = \{(x, y): \text{dist}_s(H, x, y) \geq 0 \text{ and } l_s \text{ divides } \text{dist}_s(H, x, y)\}$$

and, for $t \neq s$,

$$B_t(H) = \{(x, y): \text{dist}_t(H, x, y) \cdot l_s - \text{dist}_s(H, x, y) \cdot l_t > 0\}.$$

B_s is a semisector because, by Lemma 4.2, the set $\{(x, y): l_s(G) \text{ divides } \text{dist}_s(H, x, y)\}$ is a seminet. Moreover, by Lemma 4.5, $B_t(E)$ and $B_t(F)$ are parallel for every t. But, similarly as in Case 1,

$$\text{DOM}(H) = \text{DOM}(G) \cap \bigcap_{t=1}^{k+l} B_t(H).$$

Therefore, using Lemma 4.4, we obtain parts (a) and (b).

To prove part (d) let $c = \text{dist}_s(E, x, y)/l_s$ be the number of cycles made by A between the events corresponding to G and E. Then,

$$p_s(E, x, y) = \begin{cases} x + d_s y + 1 & \text{if } s < k, \\ 0 & \text{for } s > k. \end{cases}$$

For $t \neq s$, $p_t(E, x, y) = p_t(G, x, y) + g_t(q(E)) + c \cdot l_t$.

This finishes the proof of (a), (b), and (d). Now we will prove part (c).

Suppose that E and F are not twins. Then, for $s = \text{change}(E)$ and $r = \text{change}(F)$, we have $s \neq r$. If $q(E) \in \text{SP}(G)$ or $q(F) \in \text{SP}(G)$, then part (c) is immediate because $B_s(E)$ or $B_r(F)$ is not proper.

Thus suppose that $q(E), q(F) \in LP(G)$. Then,

$$B_r(E) = \{(x, y) : \text{dist}_r(E, x, y) \cdot l_s - \text{dist}_s(E, x, y) \cdot l_r > 0\},$$

$$B_s(F) = \{(x, y) : \text{dist}_s(F, x, y) \cdot l_r - \text{dist}_r(F, x, y) \cdot l_s > 0\}.$$

For $t < k$ and $H = E, F$ we take $a'_t = 1 - a_t$, $b'_t = b_t$, and $c_{t,H} = 1 - c_t - g_t(q(H))$. For $t > k$ and $H = E, F$, we take $a'_t = a_t$, $b'_t = b_t$, and $c_{t,H} = c_t + g_t(q(H))$. Further, let $a = l_s a'_r - l_r a'_s$, $b = l_s b'_r - l_r b'_s$, $c_E = l_s c_{r,E} - l_r c_{s,E}$, and $c_F = l_r c_{s,F} - l_s c_{r,F}$. A simple calculation gives that

$$B_r(E) \cap B_s(F) = \{(x, y) : ax + by + c_E > 0 \text{ and } ax + by + c_F < 0\}.$$

Therefore, $B_r(E) \cap B_s(F)$ is not proper. Using Lemma 4.6 we obtain part (c) because $\text{DOM}(E) \cap \text{DOM}(F) \subseteq B_r(E) \cap B_s(F)$. □

For every node E of ET(A) we define $L(E) = \text{DOM}(E) \cap L(A)$. Obviously, $L(A) = L(E_0)$ and $L(A) = \bigcup L(E)$, where the sum is taken over all leaves in ET(A).

Let us define $D_I = \bigcup_{i \in I} C_i$ for $I \subseteq \mathbb{N}$. Let E be a node of ET(A). If there exist a finite set $I \subseteq \mathbb{N}$, a 0-seminet P, and finite sets T, U such that $L(E) = (D_I \cap P) - T \cup U$, then we define $\text{rank}(E) = |I|$. If such I, P, T, and U do not exist, then rank(E) is not defined. From Lemma 4.11 it follows that there is at most one such set I, so rank(I) is well defined.

A node E of ET(A) *has property* (*) if:

(*1) $\text{DOM}(E) = S \cap P - R$;

(*2) $L(E) = (D_I \cap P - T) \cup U$,

where S is a sector, P is a 0-seminet, and R, T, and U are finite. Property (*) says that we can take the seminets in DOM(E) and $L(E)$ to be equal.

Lemma 6.5. *Let A be in normal form and E be an inner nondegenerate node in* ET(A) *with property* (*) *such that* rank(E) *is defined and* $\text{rank}(E) > 4k + 4$. *Then there is a son F of E such that* $\text{rank}(F) \geq \text{rank}(E)/(2k+2) - 2$. *Moreover, F has property* (*) *too.*

Proof. Let $L(E) = (D_I \cap P - R) \cup S$. If G is a son of E, then the events corresponding to G are caused either by one of the heads reaching the end of the block of letters

or by the change of the top symbol on the stack. Also, we may have $q(G) \in \mathrm{LP}(E)$ or $q(G) \in \mathrm{SP}(E)$. So all sons of E can be grouped into $2(k+1)$ clusters such that all nodes in every cluster are twins and nodes in different clusters are not twins. Therefore, there are numbers a_t, b_t, d_t, and e_t, $t = 1, \ldots, 2k+2$, such that, for each son G of E, $\mathrm{DOM}(G)$ is of the form

$$S(a_t, b_t, c, d_t, e_t, f) \cap P' - R',$$

for some t, seminet P', finite set R, and integers c and f. This follows from Lemma 6.2(b), (c). Let $I_t = \{i \in I : a_t < ib_t \text{ and } ie_t < d_t\}$ for $t = 1, \ldots, 2k+2$. So we divide I into sets I_t, where I_t contains those i for which the line $y = ix$ is between the lines $b_t y = a_t x$ and $e_t y = d_t x$. Clearly, if $i \in I$, then, for some t, we have $a_t \leqslant ib_t$ and $ie_t \leqslant d_t$. So there must exist a p such that

$$|I_p| \geqslant \frac{|I|}{(2k+2)} - 2$$

(we have to subtract 2 because the inequalities in the definition of I_t are sharp).

Let $F_1, \ldots, F_m$ be all the sons (and twins) of E such that, for every $j = 1, \ldots, m$,

$$\mathrm{DOM}(F_j) = S(a_p, b_p, c_j, d_p, c_p, f_j) \cap P_j - R_j,$$

for some integers c_j and f_j, seminet P_j, and finite set R_j. Since $|I| > 4k+4$, I_p is nonempty. Let $i \in I_p$. Then, $a_p < ib_p$ and $ie_p < d_p$. From these inequalities we derive that $a_p e_p < b_p d_p$. Hence, the semisectors $\mathrm{DOM}(F_i)$ are proper. Moreover, if F is a son of E from a different cluster, then F is not a twin of any F_i, that is, by Lemma 6.2, $\mathrm{DOM}(F)$ and $\mathrm{DOM}(F_i)$ are independent. Clearly,

$$\mathrm{DOM}(E) = \bigcup_{i \leqslant m} \mathrm{DOM}(F_i) \cup \bigcup_{F} \mathrm{DOM}(F),$$

where the second sum is taken on all sons of E different from all F_i. Since E has property (*), the net P appears in the description of $\mathrm{DOM}(E)$. Now we can use Lemma 4.10, from which we can derive that $P = \bigcup_{j \leqslant m} P_j$. Let r be such that $0 \in P_r$. We also define

$$Y = S(a_p, b_p, c_r, d_p, e_p, f_r),$$
$$R' = R \cup R_r \quad \text{and} \quad S' = S \cap \mathrm{DOM}(F_r),$$
$$U = D_{I_p} - Y \quad \text{and} \quad V = D_{I - I_p} \cap Y.$$

For convenience, assume that a_p / b_p and $d_p / e_p \notin I$ (it is easy to modify the proof to cover also other cases). From the choice of I_p and Y, both U and V are finite. Then,

$$\begin{aligned} L(F_r) &= L(E) \cap \mathrm{DOM}(F_r) \\ &= [(D_I \cap P - R) \cup S] \cap (Y \cap P_r - R_r) \\ &= (D_I \cap Y \cap P_r - R') \cup S' \\ &= (D_{I_p} \cap Y \cup D_{I - I_p} \cap Y) \cap P_r - R' \cup S' \\ &= (D_{I_p} - U \cup V) \cap P_r - R' \cup S' \\ &= D_{I_p} \cap P_r - (U \cup R') \cup (S' \cup V \cap P_r - R'). \end{aligned}$$

Taking $F = F_r$ we obtain the lemma. □

Lemma 6.6 (Fundamental Lemma). *Let A be a* dpda(k, l) *in normal form and E_0 the root of* ET(A). *Then $L(E_0) = D_n$ implies that $n \leq (4k+4)^{4k+1}$.*

Proof. Assume that $n > (4k+4)^{4k+1}$. If A is in normal form, then the depth of ET(A) is at most $4k$. We will find a path in ET(A) such that if $E_0, \ldots, E_m$ are all nondegenerate nodes on this path, then $m \leq 4k$, E_m is a leaf of ET(A), and, for every $i = 0, 1, \ldots, m$,

(a) E_i satisfies property (*),

(b) $\text{rank}(E_i) > (4k+4)^{4k-i+1}$.

From our assumption, $\text{rank}(E_0) > (4k+4)^{4k+1}$. Note also that E_0 satisfies property (*). Suppose that $0 < i \leq 4k$ and $E_0, \ldots, E_{i-1}$ are already found. Then,

$$\text{rank}(E_{i-1}) > (4k+4)^{4k-i+2} > 4k+4.$$

From Lemma 6.5, there is a son F of E_{i-1} satisfying property (*) and such that

$$\text{rank}(F) > \frac{(4k+4)^{4k-i+2}}{(2k+2)} - 2$$

$$= 2(4k+4)^{4k-i+1} - 2 > (4k+4)^{4k-i+1}.$$

If F is nondegenerate, then we take $E_i = F$. Otherwise we take E_i to be the first nondegenerate descendent of F. Obviously, $\text{rank}(E_i) = \text{rank}(F)$.

Let us consider now E_m. We obtain that

$$\text{rank}(E_m) > (4k+4)^{4k-m+1} > 1.$$

Therefore, $L(E_m)$ is nonempty. This implies that $q(E_m)$ must be an accepting state of A. Therefore, $L(E_m) = \text{DOM}(E_m)$. Since $\text{rank}(E_m) > 1$, there are two different numbers i and j such that $C_i \cap L(E_m)$ and $C_j \cap L(E_m)$ are infinite. From this and Lemma 4.13 we derive that $L(E_m)$ is a proper semisector. This, in turn, leads to a contradiction with the assumption of the present lemma because using Lemma 4.12 we obtain

$$\emptyset \neq L(E_m) - D \subseteq L(E_0) - D \subseteq L(E_0) - D_n. \qquad \square$$

7. Hierarchies for finite and counter automata

Theorem 7.1. *For every $k > 0$, $l \geq 0$,* DPDA$(k, l) \cap B_2 \subsetneq$ DPDA$(k+1, l) \cap B_2$.

Proof. Since $L(A) = \{1^x 2^y : (x, y) \in L(E_0)\}$ and $L_n = \{1^x 2^y : (x, y) \in D_n\}$, we have, from Lemma 6.6, that a dpda(k, l) cannot recognize L_n if $n > (4k+4)^{4k+1}$. Therefore there exists an m such that $L_m \in$ DPDA(k, l) and $L_{m+1} \notin$ DPDA(k, l). Let $L_m = L(A_1)$ for a dpda(k, l) A_1. We will show a dpda$(k+1, l)$ A_2 recognizing L_{m+1}. First, A_2 places its $(k+1)$st head on the first symbol 2. Next, it simulates the moves of A_1, except

that if A_1 moves its first head forward, then A_2 also moves the $(k+1)$st head $m+1$ cells forward. If A_1 accepts, then A_2 accepts too. Moreover, A_2 accepts when its first head reaches the first symbol 2 and the $(k+1)$st head reaches the endmarker 3 simultaneously. This gives the inequality. The inclusion is obvious. □

From Theorem 7.1 we immediately obtain the following corollary.

Corollary 7.2. *For every* $k>0$,
(1) $\mathrm{DFA}(k)\cap B_2 \subsetneq \mathrm{DFA}(k+1)\cap B_2$,
(2) $\mathrm{DCA}(k)\cap B_2 \subsetneq \mathrm{DCA}(k+1)\cap B_2$.

Theorem 7.3. *For every* $k>0$, $l\geq 0$, $\mathrm{DPDA}(k, l)\cap B_2 \subsetneq \mathrm{NFA}(2)\cap B_2$.

Proof. If $L\in \mathrm{DPDA}(k, l)\cap B_2$, then the set $\{(x, y): 1^x 2^y \in L\}$ is semilinear. This follows from the results in [20, 23]. But every such language in B_2 can be recognized by some nfa(2) (see Fact 9.1). Therefore the inclusion holds. The inequality follows from Theorem 7.1. □

Corollary 7.4. *For every* $k>1$,
(1) $\mathrm{DFA}(k)\cap B_2 \subsetneq \mathrm{NFA}(k)\cap B_2$,
(2) $\mathrm{DCA}(k)\cap B_2 \subsetneq \mathrm{NCA}(k)\cap B_2$.

Obviously, $L_2\in \mathrm{NCA}(1)$. It is not difficult to prove that $L_2\notin \mathrm{DCA}(1)$. Thus (2) also holds for $k=1$.

In [42], Yao and Rivest proved that, for every $k>1$, $\mathrm{DFA}(k)\subsetneq \mathrm{DFA}(k+1)$ and $\mathrm{DFA}(k)\subsetneq \mathrm{NFA}(k)$. The analogous results for counter automata were proved by Miyano in [27]. Corollary 7.4 strengthens these results because it shows that these inequalities even hold when we consider only simple 2-bounded languages.

Theorem 7.5. *For every* $k>0$, $\mathrm{DFA}(k)\cap B_2 \subsetneq \mathrm{DCA}(k)\cap B_2$.

Proof. The proof is similar to the proof of Theorem 7.1. Let $L_m = L(A_1)$ for a dfa(k) A_1 and $L_{m+1}\notin \mathrm{DFA}(k)$. We will show a dca($k$) A_2 recognizing L_{m+1}. The heads of A_2 simulate moves of the heads of A_1. Each time the first head moves forward, A_2 increases the counter until the first head reaches the first symbol 2 on the input. Then, after each $m+1$ moves of the first head, A_2 decreases the counter. If A_1 accepts, then A_2 accepts too. Moreover, A_2 accepts when the first head reaches the endmarker and the counter becomes 0 simultaneously. □

Theorem 7.5 states that $\mathrm{DPDA}(k, 0)\cap B_2 \subsetneq \mathrm{DPDA}(k, 1)\cap B_2$. A natural problem arises here whether Theorem 7.5 can be generalized, that is, whether $\mathrm{DPDA}(k, l)\cap B_2 \subsetneq \mathrm{DPDA}(k, l+1)\cap B_2$ for $l\geq 0$. We believe that this inequality holds only for

$l=0$ but we have no proof. We also conjecture that, in general, DPDA$(k, l) \subsetneq$ DPDA$(k, l+1)$ for $l \geq 0$. A similar hierarchy for nondeterministic automata in case $k=1$ was shown in [12].

Theorem 7.6. *For every $k>1$ and $l \geq 0$ the class* DPDA(k, l) *is not closed under union, intersection and concatenation.*

Proof. From Lemma 6.6, there is an m such that $L_m \in$ DPDA(k, l) but $L_{m+1} \notin$ DPDA(k, l). Obviously, $L_{m+1} - L_m \in$ DPDA(k, l) for $k>1$. But $L_{m+1} = L_m \cup (L_{m+1} - L_m)$, so DPDA$(k, l)$ is not closed under union. Since DPDA(k, l) is closed under complement, it cannot be closed under intersection.

Observe now that, for $L, L' \subseteq 1^+2^+$, we have $L \cup L' = (L \cup \{\varepsilon\})(L' \cup \{\varepsilon\}) \cap 1^+2^+$. But DPDA$(k, l)$ is closed under intersection with regular sets, so it cannot be closed under concatenation. □

Corollary 7.7. *For each $k>0$ the classes* DFA(k) *and* DCA(k) *are not closed under union, intersection and concatenation.*

8. The main results

In this section we will show that the method used in the preceding section to construct hierarchies for multihead finite and counter automata can be extended also to pushdown automata. This is another moment in our proof where the application of bounded languages turns out to be important. Indeed, as the following lemma states, every dpda(k) recognizing a 2-bounded language can be simulated by a dpda(k) with bounded stack.

Lemma 8.1. *For every $k>0$,* DPDA$(k) \cap B_2 =$ DPDA$(k, 2k) \cap B_2$.

Proof. We prove only the $\subseteq$-inclusion, the other one is obvious. Let A be a dpda(k) such that $L(A) \subseteq 1^*2^*$. W.l.o.g. we can assume that A never pushes a symbol when all its heads are on the endmarker. We will show a dpda$(k, 2k)$ B such that $L(B) = L(A)$. Suppose that A has m stack symbols and n states. Let $l=2k$. B will use buffers X_i, Y_i, and Z_i, for $i=1, \ldots, l$, of length mn. The contents of the stack of A will be represented in the form

$$X_1 \; Y_1^{r_1} Z_1 \; X_2 \; Y_2^{r_2} Z_2 \ldots X_l \; Y_l^{r_l} Z_l,$$

where, for each i, r_i is the length of the ith segment of the stack of B. (Above we identified buffers with their contents, that is, words over some alphabet, which is defined below).

Let an *input mode of A* be a vector $(e_1, e_2, \ldots, e_k)$, where, for every t, $e_t \in \{1, 2, 3\}$ is the symbol scanned by the tth head. Then every computation of A can be divided

into $l+1$ blocks such that in the ith block of configurations A is in the same input mode. B will work in such a way that it will make a push_i and fill buffers X_i, Y_i, and Z_i only in the ith block of configurations. B will store in these buffers pairs (x, q), where x is a stack symbol of A and q is a state of A. Buffers work in a pushdown-like fashion. Denote by $\text{pop}(T)$ and $\text{push}((x, q), T)$ the operations of B on a buffer T.

Note that B can have all information that is necessary to determine the move of A since it can store the states of A in its finite control and the top symbol of the stack of A is the top symbol of the last nonempty buffer (in the order defined above).

The heads of B simulate the heads of A. Suppose that A (or, equivalently B) enters the ith block of configurations. Then $j := i-1$ and until B enters the $(i+1)$st block of configurations, it performs the following:

(1) If A makes a pop, then
- (1a) if $|X_i|>0$, then B makes a $\text{pop}(X_i)$;
- (1b) if $|X_i|=0$ and $|Z_j|>0$, then B makes $\text{pop}(Z_j)$;
- (1c) if $|X_i|=|Z_j|=0$ and $r_j>0$ (that is $\text{top}_B=j$), then B makes a pop, $Z_j := Y_j$ and $\text{pop}(Z_j)$;
- (1d) if $|X_i|=|Z_j|=0$, $r_j=0$, and $|X_j|>0$, then B makes $\text{pop}(X_j)$;
- (1e) if $|X_i|=|Z_j|=|X_j|=0$ and $r_j=0$, then $j := j-1$ and B repeats the whole procedure.

(2) If A makes $\text{push}(x)$ in state q, then B checks whether there exists a $t \leq |X_i|$ such that $X_i(t)=(x, q)$. If such t does not exist, then B makes $\text{push}((x, q), X_i)$. Suppose that such a t exists. Then A must have entered a cycle and from this moment A will steadily increase the stack until the input mode changes. B will remember the growth of the stack during the cycle in Y_i and the number of cycles as r_i. First, B makes the following moves:

$$X_i := X_i[1, 2, \ldots, t-1], \quad Y_i := X_i[t, t+1, \ldots, |X_i|],$$

$$Z_i := (x, q) \quad \text{and} \quad \text{push}_i.$$

Now B simulates A using Z_i as a buffer:
- if A makes a pop, then B makes $\text{pop}(Z_i)$;
- if A makes $\text{push}(y)$ in state p and $(y, p) \neq (x, q)$, then B makes $\text{push}((y, p), Z_i)$;
- if A makes $\text{push}(x)$ in state q, then B makes push_i (that is, $r_i := r_i+1$) and $Z_i := (x, q)$. □

Theorem 8.2. *For every* $k>0$, $\text{DPDA}(k) \cap B_2 \subsetneq \text{DPDA}(k+1) \cap B_2$.

Proof. From Theorem 7.1 and Lemma 8.1 we obtain that

$$\begin{aligned}\text{DPDA}(k) \cap B_2 = \text{DPDA}(k, 2k) \cap B_2 &\subsetneq \text{DPDA}(k+1, 2k) \cap B_2 \\ &\subseteq \text{DPDA}(k+1, 2k+2) \cap B_2 = \text{DPDA}(k+1) \cap B_2.\end{aligned}$$ □

Theorem 8.3. *For every* $k>0$, $\text{DPDA}(k) \cap B_2 \subsetneq \text{NFA}(2) \cap B_2$.

Proof. Immediate from Theorem 7.3 and Lemma 8.1. □

As a corollary we obtain the solution of two open problems about multihead pushdown automata.

Corollary 8.4. *For every* $k>0$,
(1) $\mathrm{DPDA}(k) \subsetneq \mathrm{DPDA}(k+1)$,
(2) $\mathrm{DPDA}(k) \subsetneq \mathrm{NPDA}(k)$.

Note that part (2) follows from Theorem 8.3 only for $k>1$. If $k=1$, this result is already known. It can even be strengthened to the form $\mathrm{DPDA} \cap B_2 \subsetneq \mathrm{CFL} \cap B_2$. The proof is left to the reader.

The corollary below follows directly from Theorem 7.6 and Lemma 8.1.

Corollary 8.5. *For each* $k>0$ *the class* $\mathrm{DPDA}(k)$ *is not closed under union, intersection and concatenation.*

9. Final remarks

We have applied our technique to multihead deterministic finite, counter, and pushdown automata. The question is whether there are other applications of our method. It is not hard to see that only slight changes in the proof would yield analogous results for so-called simple multihead automata [4, 25]. However, it is not likely that other applications are possible. The reason it that we require that the considered languages have the semilinear property and that they are defined by deterministic machines.

Our method cannot be applied to nondeterministic automata. This follows from the following, easily proved fact.

Fact 9.1. *For every* $n, k>0$, $\mathrm{NPDA}(k) \cap B_n \subseteq \mathrm{NFA}(n) \cap B_n$.

Thus, the classes $\mathrm{NPDA}(k) \cap B_n$ do not form an infinite hierarchy with respect to the number of heads. The same goes for finite and counter automata. We believe, however, that such a hierarchy exists when we let n depend on k.

The other drawback of our technique is that it is not fully effective, that is, we do not determine which L_m separates k-head and $(k+1)$-head automata languages, we only prove that it exists. The problem is to find the maximum n such that $L_n \in \mathrm{DPDA}(k, l)$. We only know that this n is much greater than one would expect. For example: $L_2 \in \mathrm{DFA}(2)$ and $L_{17} \in \mathrm{DFA}(3)$! (This result was communicated to me by B. Monien and attributed to a group of his students.)

In conclusion we list some open problems related to the ones investigated in our paper.

Problem 1: Is $\mathrm{NPDA}(k) \subsetneq \mathrm{NPDA}(k+1)$ for $k > 0$?

Problem 2: Does there exist a language $L \in \mathrm{NFA}(2)$ such that $L \notin \mathrm{DPDA}(k)$ for any k?

Problem 3: Is $\mathrm{DPDA}(k) \cap B_2 = \mathrm{DCA}(k) \cap B_2$ for every $k > 0$? This is true for $k = 1$ and the author believes that the equality holds for any k.

Problem 4: Consider sensing one-way automata, that is, such automata that are able to detect coincidences of the heads. Is it possible to show analogous separation results for sensing automata using 1-bounded languages? The author conjectures that this is true.

Acknowledgment

This paper is based on my Ph.D. Dissertation written during my studies at the Institute of Mathematics, Polish Academy of Sciences. I am grateful to Prof. A. Salwicki and Dr. W. Rytter for their help and comments. I would like also to express my gratitude and appreciation to the referees for pointing out numerous errors in previous versions of this paper.

Note added in proof

Problem 1 has recently been solved in: M. Chrobak and M. Li, $k+1$ heads are better than k for PDA's, *Proc. 27th IEEE FOCS* (1986) 361-367.

References

[1] M. Chrobak, Variations on the technique of Duris and Galil, *J. Comput. System Sci.* **30** (1985) 77-85.

[2] S.A. Cook, Characterizations of pushdown machines in terms of time-bounded computers, *J. Assoc. Comput. Mach.* **18** (1971) 4-18.

[3] P. Ďuriš and Z. Galil, Fooling a two-way automaton or one pushdown store is better than one counter for two-way machines, *Theoret. Comput. Sci.* **21** (1982) 39-53.

[4] P. Ďuriš and J. Hromkovič, One-way simple multihead finite automata are not closed under concatenation, *Theoret. Comput. Sci.* **27** (1983) 121-125.

[5] R.W. Floyd, Review 14, 352 of [37], *Comput. Rev.* **9** (1968) 280.

[6] A.R. Fredman and R.E. Ladner, Space bounds for processing contentless inputs, *J. Comput. System Sci.* **11** (1975) 118-128.

[7] Z. Galil, Some open problems in the theory of computation as questions about two-way deterministic automaton languages, *Math. Systems Theory* **10** (1977) 211-228.

[8] S. Ginsburg, *The Mathematical Theory of Context-Free Languages* (McGraw-Hill, New York, 1966).

[9] S. Ginsburg and E.H. Spanier, Bounded regular sets, *Proc. Amer. Math. Soc.* **17** (1966) 1043-1049.

[10] S. Ginsburg and E.H. Spanier, Bounded Algol-like languages, *Trans. Amer. Math. Soc.* **113** (1964) 333-368.

[11] J. Gray, M. Harrison and O.H. Ibarra, Two-way pushdown automata, *Inform. and Control* **11** (1967) 30-70.

[12] S.A. Greibach, An infinite hierarchy of context-free languages, *J. Assoc. Comput. Mach.* **16** (1969) 91–106.
[13] M.A. Harrison and O.H. Ibarra, Multi-tape and multi-head pushdown automata, *Inform. and Control* **13** (1968) 433–470.
[14] J. Hartmanis, On non-determinacy in simple computing devices, *Acta Inform.* **1** (1972) 336–344.
[15] J. Hartmanis and L. Berman, On tape bounds for single letter alphabet language processing, *Theoret. Comput. Sci.* **3** (1973) 213–224.
[16] F.C. Hennie, One-tape off-line Turing machine computations, *Inform. and Control* **8** (1965) 553–578.
[17] J. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).
[18] J. Hromkovič, Closure properties of the family of languages recognized by one-way two-head deterministic finite state automata, *Proc. 10th MFCS*, Lecture Notes in Computer Science **118** (1981) 304–313.
[19] J. Hromkovič, One-way multihead deterministic finite automata, *Acta Inform.* **19** (1983) 377–384.
[20] O.H. Ibarra, A note on semilinear sets and bounded-reversal multihead pushdown automata, *Inform. Process. Lett.* **3** (1974) 25–28.
[21] O.H. Ibarra, Characterizations of some tape and time complexity classes of Turing machines in terms of multihead and auxiliary stack automata, *J. Comput. System Sci.* **5** (1971) 88–117.
[22] O.H. Ibarra, On two-way multihead automata, *J. Comput. System Sci.* **7** (1973) 28–37.
[23] O.H. Ibarra and C.E. Kim, A useful device for showing the solvability of some decision problems, *J. Comput. System Sci.* **13** (1976) 153–160.
[24] O.H. Ibarra and C.E. Kim, On 3-head versus 2-head finite automata, *Acta Inform.* **4** (1975) 193–200.
[25] K. Inoue, I. Takanami, A. Nakamura and T. Ae, One-way simple multihead finite automata, *Theoret. Comput. Sci.* **9** (1979) 311–328.
[26] M. Li, Lower bounds by Kolmogorov complexity, *Proc. 15th ICALP*, Lecture Notes in Computer Science **194** (1985) 383–393.
[27] S. Miyano, A hierarchy theorem for multihead stack-counter automata, *Acta Inform.* **17** (1982) 63–67.
[28] S. Miyano, Remarks on multihead pushdown automata and multihead stack automata, *J. Comput. System Sci.* **27** (1983) 116–124.
[29] B. Monien, The LBA-problem and the deterministic tape complexity of two-way counter languages over a one-letter alphabet, *Acta Inform.* **8** (1977) 371–382.
[30] B. Monien, Transformational methods and their application to complexity problems, *Acta Inform.* **6** (1976) 95–108; Corrigenda: *Acta Inform.* **8** (1977) 383–384.
[31] B. Monien, Two-way multihead automata over a one-letter alphabet, *RAIRO Inform. Théor.* **14** (1980) 67–82.
[32] B. Monien and I.H. Sudborough, The interface between language theory and complexity theory, in: R.V. Book, ed., *Formal Language Theory* (Academic Press, New York, 1980) 287–324.
[33] W.J. Paul, On-line simulation of $k+1$ takes by k tapes requires nonlinear time, *Inform. and Control* **53** (1982) 1–8.
[34] W.J. Paul, N. Pippenger, E. Szemerédi and W. Trotter, On determinism versus non-determinism and related problems, *Proc. 24th IEEE FOCS* (1983) 429–438.
[35] T.F. Piatkowski, *N*-head finite state machines, Ph.D. Dissertation, University of Michigan, 1963.
[36] A.L. Rosenberg, Nonwriting extensions of finite automata, Ph.D. Dissertation, Harvard University, 1965.
[37] A.L. Rosenberg, On multihead finite automata, *IBM J. Res. Develop.* **10** (1966) 388–394.
[38] J.I. Seiferas, Relating refined space complexity classes, *J. Comput. System Sci.* **14** (1977) 100–129.
[39] J.I. Seiferas, Techniques for separating space complexity classes, *J. Comput. System Sci.* **14** (1977) 73–99.
[40] I.H. Sudborough, One-way multihead writing finite automata, *Inform. and Control* **25** (1976) 1–20.
[41] I.H. Sudborough, Some remarks on multihead automata, *RAIRO Inform. Théor.* **11** (1977) 181–195.
[42] A.C. Yao and R.L. Rivest, $k+1$ heads are better than k, *J. Assoc. Comput. Mach.* **25** (1978) 337–340.